Change Report

"Mathochist Studios" Cohort 4, Team 11

Euan Cottam
Charlie Thoo-Tinsley
Harri Thorman
Will King
Zach Moussallati
Aiden Turner
Marcus Williamson
Joshua Zacek

Part a)

Planning Changes

Whilst maintaining originals as a baseline for comparing at a later time, we commenced the process by copying over the other team's Assessment 1 PDFs into shared Google Docs. With a view to covering all inherited documentation and code for Assessment 2, each member of the team was assigned deliverables to review (requirements, architecture, methods, risks, implementation).

Tools for Making/Tracking Changes

With pre-determined colours/labels, Google Docs suggestions, comments, and highlighting of text were used to distinguish unchanged content, content to revise and gaps, and where new content was required. Whilst implementation/website were maintained in GitHub, each change was shown through commits that reference the relevant inherited deliverable sections and requirement IDs – as such supporting traceability.

Conventions for Combining Both Teams' Work

On each deliverable, we firstly compared our own Assessment 1 deliverable, and the chosen team's version, and then summarised which section were stronger in each document in a separate Google Doc comparison. By retaining stronger sections verbatim where appropriate, improving weaker sections based on our prior work and other team's material, and adding new material where both versions lack, we then drafted a "super deliverable" for Assessment 2.

Reviewing and Approving Changes1

Written deliverable changes were discussed via Google Docs comments, in meetings and in WhatsApp. Comments that remained unresolved were then used as a checklist before finalising. As for the code, any merge/pull requests were used such that at least one team member would look over the new or modified classes. The reviewer was to check the consistency with the updated architecture, as well as requirements, before being approved. Before final submission, a team member was to do a final check to make sure that all PDFs, code and material were correctly updated, and linked from the Assessment 1 website, with major changes being clearly identified.

Changes to Requirements

Original Document:
https://drive.google.com/file/d/1b5qR4DtEDMKsCl5eQtCTsR3tOAv6N4cb/view

Updated Document:
https://drive.google.com/file/d/1HtJEzsI8IrS2S5EmgVj1nFRsxhvZgoUD/view?usp=drive_link

There were a few issues we felt we had to change with the previous team's requirements deliverable. We felt as though it needed a slight overhaul.

All requirements were previously labelled with numerical IDs. This can make it confusing when the requirements are referenced in a different context (e.g. in implementation). We changed all the numerical IDs into more meaningful IDs (e.g. UR1.05 to UR_UNIVERSITY_ACCURATE).

We grouped all of the User Requirements in order of priorities, as before they were all mixed. We felt that this was essential as it makes it much clearer to see which requirements are essential and must be implemented, so as to make sure they are not missed or overlooked. Some requirements were also labelled as multiple priorities (e.g. shall/should), this can cause confusion when choosing which requirements should be prioritised. We made sure all User Requirements have only one priority.

The User Requirements were previously not organised into hierarchies.
We sorted them into hierarchies depending on which component of the game each user requirement was related to (E.g. gameplay, menu, etc). We felt this would make it easier/more efficient when sorting through many user requirements.
The System requirements were organised into hierarchies, however we felt some of the groupings were slightly confusing. We regrouped all the system requirements into groupings we felt were more coherent and understandable.

We also felt as if many of the User Requirements could be combined into one slightly broader User Requirement, then there could be multiple system requirements solving this one larger User Requirement. We thought this would be a lot more concise, efficient and easier to understand, as it reduces the clutter in requirement tables, whilst keeping the same solutions to the User Requirements.
An example of this would be the previous User Requirements UR1.03, UR1.06 and UR1.16, each with their own corresponding System Requirements. We combined these requirements into one "UR_CASUAL_FRIENDLY", and then have system requirements, FR_EASY_MOVEMENT_CONTROLS, NFR_CASUAL_FRIENDLY_DIFFICULTY and NFR_CLEAR_BOUNDARIES all correspond to this one broader User Requirement.

In the previous teams Non-Functional Requirements table, we found that some of their fit criteria columns were imprecise or too vague, and it required subjective judgement to gauge whether or not the fit criteria was made or not.
For example, "Game should be appropriate for all ages, with no harsh effects or features".
No guideline is listed to verify what is considered a harsh feature, there could be conflicts between clients and development team on whether something is considered harsh or not appropriate for all ages.
We made sure each and every requirement had an objective way of verifying whether the fit criteria was met for each requirement. (E.g. if the game adheres to the BBFC PG rating guidelines stated on their website, then NFR_NO_GRAPHIC_CONTENT can be considered met).

We felt some of the requirement definitions were quite verbose and lengthy, this can make it more difficult for the team to read when dealing with many requirements.
We made some descriptions more concise and easier/more efficient to understand.
E.g. Previous description for NFR_1.12: “Making the volume adjustable and mutable caters to all player’s preferences. Which is very important in consistently providing an enjoyable and comfortable player experience.” was reduced to “The game can include a volume slider that lets users adjust or mute the sounds from the game.”

We also added some new Requirements, both the two new ones added in assessment 2 (leaderboard and achievements), along with some extra requirements that we felt the previous team was missing before (e.g. UR_SCREEN_SCALABILITY).

Part a) was slightly changed to reflect the changes stated above to the presentation of the requirements tables (e.g. priority sorting, mentioning requirements hierarchies, requirement IDs etc).

Changes to Architecture

Original Document:
https://drive.google.com/file/d/1pvDhoANljp0z9YtdjMN0SxkNib2iyxAI/view

Updated Document:
https://drive.google.com/file/d/1hNc_xgVSvAdL0_IOj1V7jzJJD3WTx7SM/view?usp=drive_link

To start, the Architecture document initially didn't contain an adequate statement going over the design languages and tools used. To rectify this, we included a brief statement of the tools we used, being UML (diagrams), PlantUML (language for creating diagrams), and PlantText (text editor for language).

The original document's explanation of "static v. dynamic" entities was cluttered and unclear. We felt it was necessary to rephrase this section to make it easier to read and understand. On the topic of "static v. dynamic" entities, the UML diagrams for these in the original document were inconsistent. To resolve this, we recreated these diagrams fixing the inconsistencies (i.e. missing Entities on detailed UML diagrams compared to abstract UML diagrams).

We felt that the original documents Architecture ("static v. dynamic") was flawed and lacked real meaning. As a result, we decided to use a completely new Architecture for our entities ("enemy v. interactable v. legacy"). This is an excerpt of our newer Architecture explained:

"At later stages, we decided this architecture model for our entity was unfit for purpose; We didn't find stationary and moving entities to be a meaningful distinction. We decided to use an Object-Oriented architecture with several separate types of entities: InteractableEntity, Enemy, and XAxisSlidingEntity (legacy).

We kept XAxisSlidingEntity because there were some entities from earlier stages of development that we wanted to continue using in their current form, and refactoring them to use Enemy would've been too time consuming. In our final architecture, only one entity uses this legacy Entity type.

The remaining two Entity types have two important distinctions: Enemy is triggered by a player collision with a specified radius defined by EnemyAI, while InteractableEntity is triggered by a player key press within a specified radius of the InteractableEntity. Additionally, Enemy uses EnemyAI to handle movement, whereas InteractableEntity is static by default i.e. it must define its own movement. EnemyAI system makes this easy to implement and modular i.e. you can use one EnemyAI with multiple Enemy entities. This is particularly useful for EnemyAIs with complicated movement algorithms, allowing for decomposition.

You can see these differences reflected in Figures 5 and 6, where InteractableEntity has its radius defined in the main interface (interactionRadius: float), while Enemy has its radius defined as a private attribute in an EnemyAI. Additionally, you can see the "static-as-default" behavior of InteractableEnemy in entities such as BirdSeed, which have movement-related methods and attributes. Although there are InteractableEntity entities which can move i.e. Bus, but they need to define their own movement. In contrast, Enemy passes this responsibility to EnemyAI."

To accompany the new Architecture, we created several new UML diagrams (Figure 5 through 7).

Additionally, we felt that the original way regions operated was restricting, further backed up by our user testing. As a result, we changed regions to be more fluid i.e. ability to go back and forth between regions. We also added further content to each region, including required steps for game completion, to encourage exploration and back-tracking. We created a new UML state diagram (Figure 10) to show the revised win condition. Below is an excerpt of our revised region model:

“Figure 8 portrays how each region followed on from each other initially and the different timepoints of gameplay. Every region had a unique interface based on a university landmark (NFR_UNIVERSITY_ACCURATE_LOOK). The shop is a separate level which has purchasable items dispersed around the level. Simply colliding with the items will buy them.

At later stages, we decided that this model, while on the right track, was too restrictive, especially after conducting user testing, which indicated to us that the “forward-exclusive” directionality frustrated users.

As a result, while keeping the “region” model, we decided to make the regions bi-directional, so that you could go back and forth. Combined with additional content in each region, modification of the win condition (see next section), and a more “maze-like” experience, user exploration was incentivized, enhancing the gameplay experience.”

Changes to Method Selection & Planning

Original Document:
https://drive.google.com/file/d/1kvnNaF7vWzXJBQs-Mfe-2aBUGakdRnww/view

Updated Document:
https://drive.google.com/file/d/1EvYsmYE3an_obGvnIrHLezRO2MTm1g0j/view?usp=drive_link

Part (a)

For this portion of the deliverable, the initial description of Agile Methodology & its suitability for the project was left unchanged because our team was using the same methodology. However, the breakdown of how sprints were structured was changed to better match our team's organisation; for example, how each team implemented testing & CI verification varied, so this was adjusted. Additionally, collaboration tools were added to align with the deliverable specification. Development tools and justification had to be added to align with the assessment document & previous deliverables' lack of them.

Part (b)

Early on, during a review of the previous team's work, it was found that certain problems with the project were most likely caused by the team's previous structure. For example, their decision to develop each room independently resulted in a lot of redundant code that needed to be refactored. Additionally, our workflow preferences motivated the decision to adopt a different organisational structure than the previous team. However, due to the structure of the previous team's organisational report, it was very difficult for our team to adapt it to fit our chosen organisation's structure. Therefore, the decision was made to use a different organisational structure. The previous team's organisation has been kept in documentation, reworded, in order for clarity & to give a clearer view of the whole project.

Part (c)

This portion of the deliverable has only been appended to as a whole new plan has needed to be implemented due to the project only being followed on from.

Changes to Risk Assessment & Mitigation

Original Document:
https://drive.google.com/file/d/1Bg2mYbbsqG9NKQiwxwWKyA25hvXxRnWG/view

Updated Document:
https://drive.google.com/file/d/1I9EatJHRunmLQ4jRGuwIUKE41kjn7Tn9/view?usp=drive_link

Generally, the ‘Risk Assessment and Mitigation,’ deliverable that we had taken followed a quite straightforward plan of action. The risk management strategy was similar to our first deliverable that was submitted and followed a simple four-step approach: identification, analysis, planning and monitoring. There were however a few changes that needed to be made.

The first thing that was missing was the fact that no colour coding was used in the risk register. As mentioned in the lectures, the use of colour coding is intended to help more easily identify and quantify higher-severity risks that need to be monitored.

The second thing that needed changing is more subjective in nature. It relates to renaming the identification of each risk in the risk register. Once again referring to the lectures, unlike requirements we were expected to give each risk a numeric identifier. Although giving each risk a unique name makes it clear what the risk is, when it comes to referencing risks in the register it can be hard to navigate, particularly since the register is not organised in a particular way.

There were four distinct types of risk we were told to identify: Project, Product, Project and Product, and Business. The absence of ‘Project and Product’ risks can be forgiven since the group may have decided that risks leaned towards one or the other; however, there were no business risks identified. In a professional environment this could prove problematic, as it shows that the previous team was only thinking about the short-term concerns but not long-term goals. An example business risk which we have added at the end of the risk register is the risk of misinterpreting client expectations for events.

Many of the risks that are identified in a risk register are usually specific to each group project; however, we felt that there were some risks that were missing that are universal amongst all the groups. A key one that stood out was the risk of dependencies becoming incompatible or unavailable. An updated library may conflict with the project and cause a significant hindrance to the project.